Analysing Onion Routing
Bachelor-Thesis

Steffen Michels

June 22, 2009

Abstract

Although methods for reaching security goals such as *secrecy*, *integrity* and *authentication* are widely used in the Internet, there is no widely-used solution providing *anonimity*. Among all existing approaches to provide anonimity *onion routing* is a very promising one.

In this paper the security goals concerning anonymity are given and it is shown that onion routing can in principle provide solutions for the most important ones. Still there are many open problems and solutions trying to solve some of them, like *universal re-encryption*, introduces new problems by themselves. The only solution that is mature enough and also has enough participants to provide a reasonable level of anonymity is TOR. But still anonymity provided by it is far from being perfect.

1 Introduction

Nowadays methods for reaching security goals such as *secrecy*, *integrity* and *authentication* are widely used in the Internet. This is reached by using encryption, digital signatures and certificates. But even when using an encrypted connection with an authenticated server, it is still possible to determine who is communicating with whom by listening to the traffic. But *anonymity* is also a very important part of privacy and solutions for this are needed.

Why is anonymity important? For people who are living in countries, in which it is prohibited to communicate with people having opinions that are against the ideas of the government or visiting websites which contain such ideas, it might be dangerous to do so. For these people anonymity is a very important issue. But also people who live in democratic countries might not want other people to know what they are doing in the Internet. People like journalists or companies may have the need for anonymity, too.

Despite of this there is no widely used solution to achieve anonymity, although there are some attempts. One of these is *onion routing* which is a technique for anonymous communication over a public network. With this technique messages travel from source to destination via a sequence of onion routers which routes the messages via an unpredictable path.

The goal of this paper is to make a security analysis of onion routing. The most important issue here is how strong the anonymity is. But this question cannot simply be answered because it depends on the situation. There are different kinds of applications and also some different ways how to define what anonymity means in certain circumstances. Also there are a number of different approaches how to use the basic idea of onion routing, which might provide different levels of anonymity for different cases. So I focus on the differences between these variants.

First in Section 2 a security analysis of anonymous communication is given and existing solutions are given. After this the idea behind onion routing and possible attacks are given in Section 3. Then a number of different approaches to implement onion routing are discussed. It is discussed for which situation these approaches try to provide which kinds of anonymity, what the possible attacks are and which vulnerabilities of previous variants they try to fix. The approaches which are discussed are: the first implementation (Section 4), TOR (Section 5) and approaches based on *Universal Re-Encryption*

(Section 6). In Section 7 the variants are compared and the anonymity that can be reached with onion routing in general is discussed together with the open problems. Finally conclusions are drawn in Section 8.

2 Anonymity

Here a security analysis of anonymous communication is done.

2.1 Stakeholders and Assets

Stakeholders of the system are all parties who want to communicate anonymously for one of the reasons already discussed in the introduction. But there are different kinds of anonymity that someone might want to achieve. For example someone might want to communicate with someone and authenticate to her, but no external observer should be able to know about this communication. It is also possible that a person wants to access information anonymously and that not even the server knows her identity. Another possibility is that someone wants to publish information anonymously.

So the primary asset depends on what the stakeholder wants to achieve and can consequently be the information who is communicating with whom, who accessed which information or who published with information. Of course, secret keys are also assets of the system.

There are also different levels of anonymity that the stakeholder might want to achieve. Someone could only want to make traffic analysis difficult such that it is not feasible to make a profile of her communication behaviour, but does not mind if single connections can be traced with much effort. But also there are persons who really wants to keep the fact that they are communicating with one other specific person secret.

2.2 Attacker Model

There are also different kinds of attackers who want to achieve different goals. One could try to make profiles of a large group of people for commercial reasons. But also one can try to analyse the communication data of a single person to take advantage from that or to see if the person does something which is prohibited. Also an attacker might want to verify if two persons are communicating or not.

The possibilities of an attacker have to be taken into account, too. There might be a global observer who can observe all traffic between all nodes of the anonymous network. Other assumptions about the attacker can be that she can only observer a small part of the network or only has one node of the network under control.

2.3 Security Requirements

The information that should be protected can be seen as a kind of confidential data and so the main security requirement is the confidentiality of this information. Of course there are also some derived security goals such as the confidentiality of the keys used. The goal of solutions for anonymous communication is not to provide *confidentiality* or *integrity* of the messages send. For this users can run protocols providing this on top of the anonymous connection.

In the following only the primary security requirements dealing with anonymity are given, assuming the situation that Alice establishes a connection to communicate with Bob. The first security requirements focuses only on the involved parties and not on an external attacker:

SR1 Alice does not know the identity of Bob.

SR2 Bob does not know the identity of Alice.

The first requirement is important if someone wants to publish information but wants to stay anonymous. For example Bob wants to run a web-server but people connecting to it cannot get to know the real address of it but only connect to it with help of a pseudonym what gives no information about the real identity. The case that someone wants to communicate with someone without revealing her identity is reflected by the second requirement.

More important is the situation in which there is an external attacker who wants to compromise anonymity. In the following requirements this attacker is called Eve.

SR3 Eve cannot get to know with whom Alice is communicating.

SR4 Eve cannot get to know with whom Bob is communicating.

SR5 Eve cannot confirm whether Alice and Bob are communicating.

There is an essential difference between the first two and the last requirement. In the first case Eve starts with one person and should not be able to get the information to whom this person connects (*SR3*) or from whom the person receives connections (*SR4*). In the case of *SR5* Eve focuses on a pair of persons, for instance because she has an evidence that those persons are communicating, and tries to confirm that. In practise this last requirement is very difficult to achieve.

In some cases it is desirable to achieve *SR3* and *SR4* but not *SR1* and *SR2*. This is possible by using a protocol providing authentication on top the the anonymous connection. So Alice and Bob are mutually authenticated and know with whom they are communicating but no external party does.

A last very strong property is *steganography*, which means that it is impossible to confirm that Alice is using anonymous communication:

SR6 Eve cannot confirm that Alice is communicating via an anonymous connection.

2.4 Differences with Other Solutions

There are many different solutions for providing anonymous communication for different kinds of technologies and assumptions. Here some solutions are discussed and the assumptions for which they are built are compared to those for onion routing.

Unlike proposals for providing anonymity in for example *ISDN* networks [16] where a circuit is build and used to transport the data during one connection, onion routing is designed to be used in a public *packet-based* networks such as the Internet, where each single packet is routed from one router to the next one until the receiver is reached. So each packet has to contain not only the message but also routing information such that each router knows to which next router the packet must be send. In the most simple case the routing information is just the address of the receiver. Even if the message is encrypted such that only the intended receiver can read it, the routing information has to be readable by all routers. So someone who can read the messages processed by the routers can get to know who is communicating with whom.

In contrast to for instance Chaum's MIX networks [2], which provides anonymity for services where a large latency can be introduced such as email, onion routing is designed to work for services which require real-time communication such as *HTTP* or *SSH*. This property makes some attacks, especially timing attacks, more likely.

There are other simply designs with similar goals as onion routing. One example for this is *the Anonymizer* [1]. Here the data is send to an anonymizing proxy which then sends the request to the destination. The problem with this design is that users have to trust a single proxy. Onion routing is designed in such a way that a number of proxies are used and anonymity is reached if at least one of them is not compromised.

3 Onion Routing

In this section the goals, the principles and the possible vulnerabilities of onion routing are given.

3.1 Goals of Onion Routing

In its basic form onion routing tries to achieve *SR2*-*SR5*, although none of the existing solutions are designed for reaching the last one entirely. Some variants of onion routing also gives the possibility to provide anonymous services to others (*SR1*). The goal of onion routing is not to hide who is connected to the network (*SR6*).

The goal of onion routing is not confidentiality and integrity of the data sent via the network. Users should be aware that the confidentiality and integrity of data transfered with protocols not providing this via the Internet is never ensured and should use protocols providing confidentiality or integrity if they want to achieve it.

3.2 Design of Onion Routing

Rackoff and Simon proposed a protocol similar to onion routing for the first time [17], but they did not use the name *onion*. Later Goldschlag, Reed and Syverson introduced the name onion routing and made a first proposal for a design usable to make an implementation [10].

3.2.1 Routes

With onion routing a fixed route which is determined by the sender is used to do the routing. This is done in such a way that each router only knows the previous and the next router in the route. If someone wants to know with whom the sender is communicating she has to control all routers of the route. Even if only one router is not under her control it is impossible to figure out the recipient. However this is only true for an idealized situation, because there are many possible attacks as will be discussed later. Sending messages is done using onions. How onions are constructed is described later.

In Figure 1 an example is given in which *Alice* wants to send message m to *Bob* using three routers. *Alice* first has to construct $Onion_A$. A knows who *Alice* is but can't get to know m or who *Bob* is. A can only construct $Onion_B$ and knows that this has to be send to B. B only knows who A is and that it has to send the next onion to C. C knows m and that the intended receiver of m is *Bob*, but there is no possibility to get to know that the original sender is *Alice*.

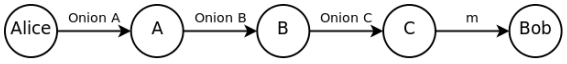

Figure 1: An Onion Routing Example

3.2.2 Onions

An onion is a recursive data-structure, encrypted with a key K_R, such that only the intended recipient R of the onion can decrypt it. K_R can be a public key of R or a symmetric key which is known by R and the constructor of the onion. The encrypted onion contains the next node R' in the route as first part. This information is readable by R after decryption. After this the onion contains another onion encrypted with a key $K_{R'}$ such that it can be decrypted by R'. The only exception is the case that R is the last node of the route. Then the onion contains the data intended for the receiver instead of another onion.

An example of a an onion with three layers in which the first parts contain the next recipient of the onion is:

$$\{B, \{C, \{Bob, data\}_{K_C}\}_{K_B}\}_{K_A} \tag{1}$$

3.2.3 Sending Messages with Onions

The main idea is that if a router receives an onion it only knows the previous router in the route and can get to know the next one, since its identity is included in the first part of the onion which is readable after decryption. It is unable to decrypt the next onion, the only thing it can do is to relay it to the next router in the route. Only the first router knows the original sender of a message and only the last one knows the intended recipient.

The example given in Figure 1 can be realized using the onion in Equation 1. Alice has to know all K_i to construct the onion. In practise it is not trivial to give all the senders access to the keys of all nodes of the network, but this is discussed later. Alice can then send the onion to A. A can decrypt the onion and knows that it must send the second part of the onion to B. Finally C gets the last onion and can send m to Bob in plaintext.

Below the whole protocol is given:

$$Alice \rightarrow A : \{B, \{C, \{Bob, m\}_{K_C}\}_{K_B}\}_{K_A} \tag{2}$$
$$A \rightarrow B : \{C, \{Bob, m\}_{K_C}\}_{K_B} \tag{3}$$
$$B \rightarrow C : \{Bob, m\}_{K_C} \tag{4}$$
$$C \rightarrow Bob : m \tag{5}$$

3.3 Attacks

There are two types of attacks: passive and active ones. Passive here means that the attacker has no own nodes in the network and cannot modify messages, but can only listen to traffic. Assuming that an attacker can listen to all the traffic might be paranoid if you consider for instance the Internet. But still a government or intelligence organizations might record a huge amount of Internet traffic or get access to particular interesting traffic, for example the communication of one person.

For active attacks the attacker modifies messages and controls a fraction of the nodes. One of the ideas of onion routing is that the user does not trust a single server but uses a route of nodes and assumes that at least one of them is not compromised. It has to be assumed that an attacker can only control a fraction of nodes such that the chance that

all nodes of one route are controlled by her is small enough. But even with only a small fraction of nodes there are some possible attacks.

The content a user sends could also possibly tell something about her identity. Examples for this are information a browser sends to a website like referrers or cookies. Note that the content may not be encrypted between the last node of the onion network and Bob. Also Bob self could be the attacker. Those attacks are outside the scope of onion routing since it does not change the content of messages itself, there are other solutions for this.

Below a number of different attacks applicable to onion routing in general are given. The list is not complete, since there is a large number of possible attacks.

3.3.1 Passive Attacks

Reconstructing Onions A simple attack, which is e.g. given in [12], is based on the fact that it can be possible to re-construct the onion a node has received before from the one that is send further to the next node after decrypting it. For example assume that an attacker records all incoming traffic for a node B for a small amount of time. B receives an onion from A and then sends the included onion o to C:

$$A \rightarrow B : \{C, o\}_{K_B} \tag{6}$$
$$B \rightarrow C : o \tag{7}$$

The attacker observing o can reconstruct $\{C, o\}_{K_B}$, because the public key of the node K_B and the destination C of the outgoing onion is known. The attacker can use the recorded incoming traffic for B to find out that B received $\{C, o\}_{K_B}$ from A. This can be done for each node until the attacker knows the entire route.

Decreasing Onion Size Another problem with the simple scheme given in the previous section is that onions become smaller and smaller while traveling along the route. This can reveal the route of a message.

Fingerprint Attacks *Fingerprints* mean characteristic file sizes, access patterns and latencies for either nodes of the network or services which are potentially used. For example a collection of website fingerprints can be used by an attacker to verify if someone is visiting one of those.

End-to-End Attacks *End-to-end attacks* are attacks where an attacker observes two potential ends of a connection and tries to confirm if they are really the ends of a connection. Those attacks try to break *SR5*.

One of such attacks would be to simply count the packets coming in and out and try to find a correlation.

A more sophisticated attack is a *timing attack*. Even if not considering the content or the size of onions an attacker can see that messages are send. This information can be used for a statistical analysis. An important factor is how much the network is used. If more users use the same nodes, a statistical analysis becomes more difficult. But still it is always possible if collecting enough data.

Of course this attack can also be used to get to know a route one-by-one instead of confirming if two persons are communicating, but of course this requires much more effort.

3.3.2 Active Attacks

Evil Nodes The most obvious attack is to run nodes of the network by yourself. But since onion routing is designed with the assumption that not all nodes are trustworthy, it is not enough to simply run one evil node. In the case that all nodes of a route are compromised, of course anonymity is broken.

In [18] probabilities for such attacks are calculated under some assumptions. If the network consists of r nodes and c are under the control of an attacker, the probability that an attacker controls the end and the begin point is c^2/r^2. Controlling the begin and end point is enough to compromise all the content and also it is very easy to confirm that source and destination are communicating as discussed before. This shows that the size of the network is essential for the anonymity provided.

Repetitive Attack One other attack is a repetitive attack. The problem is that two equal onions one node receives results in two equal onions which are send to the next node. For example assume that the attacker *Eve* is a node herself. She gets

an onion from node A and sends the next onion to node B. The next node on the route is C:

$$A \rightarrow Eve : \{B, \{C, o\}_{K_B}\}_{K_{Eve}} \quad (8)$$
$$Eve \rightarrow B : \{C, o\}_{K_B} \quad (9)$$
$$B \rightarrow C : o \quad (10)$$

Eve does not know o and consequently also not that C is part of the route. If *Eve* observes all outgoing traffic of B after she sent $\{C, \{D, o\}_{K_C}\}_{K_B}$ to this node, she can reveal C. To do this she has to send the same message to B again. This will lead the to same onion o send to C. By finding the duplicate message o in the outgoing traffic of B, *Eve* now knows the receiver of o and therefore the next node in the route which is C.

Iterated Compromise Another active attack is to compromise all nodes of a route after the onions have been send. If the attacker has one recorded onion and gets to know the private key of the receiver, she can decrypt it and reveal the next node in the route. This can be done for the whole route. There are several ways to compromise the nodes. Possibilities are to hack them all or to get physical access to them. Also a court may force the owners of the nodes to give them the keys.

3.3.3 Directory Attacks

In large scalable networks the list of nodes has to be distributed in some way. This is an interesting attacking point because manipulating the directory system in such a way that clients have different knowledge makes traffic analysis easier. Also an attacker could force clients to use nodes under her control.

4 First Version

The first idea how to implement onion routing was given by Goldschlag, Reed and Syverson [10]. To make onion routing work for the Internet without the need to change the existing infrastructure or the used applications, it is implemented to work as a *proxy server*. The disadvantage of this approach is that it works only for *application layer* protocols which are supported by the implementation and the used applications must also be able to use proxy servers. But still it is possible to provide an implementation for the most important Internet protocols, such as *HTTP* and many applications, like web-browsers, can be configured to use proxies.

4.1 General Design

The design consists of a network of routing nodes which can communicate with each other. On the Internet this is simply a *TCP* connection between each of them. For the communication they use asymmetric cryptography. So each node must be aware of at least a number of other nodes and have their public keys. In this design it is not dealt with the problem how to keep the list of nodes up to date, because it is intended to be a prototype with only a few nodes. In later versions there are solutions how to distribute the list of nodes and their keys.

If Alice wants to communicate with Bob she simply uses the entry point into the network as a proxy. The connection between Alice and the proxy is unsecured and reveals all information about with whom Alice is communicating. So the connection has to be done via a trusted network. The simplest solution would be that the software Alice uses and the first node are running on the same computer. The first router chooses a route through the onion network and sets up a *virtual circuit*. Details about virtual circuits are given in the next section. Then messages from Alice are send through the route to the last node. The communication between the nodes does not reveal with whom Alice is communicating, as described in Section 3.2. The last node then makes a connection to Bob and sends the messages to him. Note that this last connection is not encrypted (if the protocol Alice uses to communicate with Bob does not do that) and reveals the messages. But according to the goals of onion routing (Section 3.1) this is not a problem because an attacker cannot determine the originator of the messages.

Answers from Bob are send back to Alice through to route the other way around. No-one, not even Bob, knows the destination of the messages. He only knows the node where the message left the onion network.

4.2 Virtual Circuits

In the implementation onions are not used to transport data but only to build virtual circuits which uses repeated symmetric encryption. This is because it is much more efficient than using onions, which requires asymmetric cryptography, and also the same anonymity as for onions can be achieved. In this section it is described how to build these circuits using onions and how they are used to send data.

4.2.1 Creating Circuits

To create a new circuit the first node has to construct an onion which describes a fixed sequence of nodes and sends it to the first node of the chosen route. The sequence of nodes is encoded by nested onions.

Each node can decrypt the onion it receives. To prevent repetitive attacks each onion has an expiry time. Expired onions are not accepted. Additionally a list of all not expired onions has to be kept by each node and for each onion it has to be checked if it was received before. If the onion is valid the node chooses a virtual circuit identifier. Each circuit has different identifiers for each connection between two nodes in the route. The node stores this identifier together with two *cryptographic function/key pairs* which are included in the onion and are later used to send data through this circuit, as described in the next section. If the node is not the last one in the route the payload, which then contains another encrypted onion, is send to the next node. To keep the size of all messages fixed random padding is added to each onion. Because the payload is encrypted no one, except the last node of the route, can tell where the padding begins. This is important because otherwise the decreasing size of the onion would reveal too much information.

In detail an onion used for this looks like this:

$$\{exp\_time, n\_hop, F_f, K_f, F_b, K_b, payl\}_{PK_x} \quad (11)$$

The fields have the following meaning:

- *exp_time*: the time the onion expires (used to prevent replaying)
- *n_hop*: the next router in the circuit
- F_f, K_f: a cryptographic function/key pair which is used for data traveling in forward direction
- F_b, K_b: a cryptographic function/key pair which is used for data traveling in backward direction
- *payl*: another onion (with added padding) or only random padding if the receiver is the last router in the circuit
- PK_x: the public key of the receiver

The constructed circuit can then be used to exchange messages between Alice and Bob. If it is not needed any more or an error occurred, the circuit can be destroyed by any node by sending a special *destroy-message*.

4.2.2 Sending Data

If the Alice wants to send data she first splits the incoming packets into chunks such that they can be send inside the payload of messages. Then all cryptographic operations are applied to this payload in reverse order and it is send as a *data-message* together with the correct circuit identifier.

If a router in the middle of the circuit receives a data-message it can look op the cryptographic function/key pair for the corresponding circuit identifier. If data is traveling in the forward direction the first function/key pair is used. For the other case (data is traveling in the backward direction) the second function/key pair is used. After the *payload* is decrypted it is send to the next node together with the circuit identifier for the connection between the current and the next node, which is stored in the current node's table.

If the last router decrypts the payload it actually has the data in plaintext and it can be forwarded to the responder.

This also works the other direction around from the responder's to the initiator's proxy in a symmetric way.

4.2.3 Example

If for instance Alice wants to communicate with Bob using the onion nodes A, B and C the resulting virtual circuit looks like in Figure 2. The connection between Alice and A and the connection

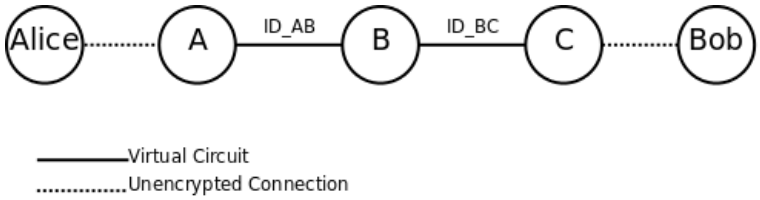

Figure 2: Virtual Circuit Example

between C and Bob is unencrypted. The circuit has two identifiers in this case for each connection between nodes.

The information stored in the nodes for this circuit are given in Table 1. The initiator of the circuit A knows all the function key pairs used in the circuit. Nodes in the middle of the circuit (here only B) have to know the next hop, the next identifier ($ID2$) and the function/key pair to apply (forward) for each possible incoming circuit identifier ($ID1$). The last node in the route C does not need a second identifier because the message is send to the next destination Bob in plaintext.

ID1	next	ID2	forward	backward
...	...	...	...	...
-	B	ID_{AB}	P_{fB}, P_{fC}	P_{bB}, P_{bC}
...	...	...	...	...

(a) Table of A

ID1	next	ID2	forward	backward
...	...	...	...	...
ID_{AB}	C	ID_{BC}	P_{fB}	P_{bB}
...	...	...	...	...

(b) Table of B

ID1	next	ID2	forward	backward
...	...	...	...	...
ID_{BC}	Bob	-	P_{fC}	P_{bC}
...	...	...	...	...

(c) Table of C

Table 1: Example Tables of Onion Nodes

In the following example $P_x(m)$ means that the function/key-pair P_x is applied to m and $P_x^{-1}(m)$ means that its inverse it applied to m. If $Alice$ wants to send a message m through this virtual circuit she just sends it to A in plaintext. A then applies the inverse of all function key-pairs to m in reverse order and sends it to the next node together with the correct circuit identifier:

$$Alice \rightarrow A : m \tag{12}$$

$$A \rightarrow B : \{ID_{AB}, P_{fB}^{-1}(P_{fC}^{-1}(m))\}_{K_B} \tag{13}$$

B can look up the corresponding function/key pair for ID_{AB}, compute $P_{fB}(P_{fB}^{-1}(P_{fC}^{-1}(m))) = P_{fC}^{-1}(m)$ and send this to C together with the second identifier stored in the table:

$$B \rightarrow C : \{ID_{BC}, P_{fC}^{-1}(m)\}_{K_C} \tag{14}$$

Finally C can compute $m = P_{fC}(P_{fC}^{-1}(m))$ and send it to Bob:

$$C \rightarrow Bob : m \tag{15}$$

For data sent from Bob to Alice the backward function/key pairs are used and A can get the plaintext by applying the inverse of all operations.

4.3 Reply Onions

For some applications it is desirable that the receiver is able to send an answer after the original circuit is broken. This is for example the case for e-mails. For this *reply onions* are used. With these onions a new virtual circuit is established after the original one is broken. The reply onion is send by the last node of the original circuit to the initiator which then can establish a new circuit. This circuit behaves as each virtual circuit once it is set up. For this the innermost payload contains all information which is determined by the initiator in the normal case, that is all function/key pairs.

4.4 Vulnerabilities

Here for the attacks described earlier it is discussed how likely they are with this design. Also some

attacks not directly connected to onion routing but only to the implementation are discussed.

4.4.1 Passive Attacks

Listening on Connection to First Node With designing the begin of a route as a proxy it is assumed that the connection from the initiator to the proxy is done via a trusted network. This obviously introduces new vulnerabilities. The most secure variant would be that the proxy is running on the same computer than the initiator's software.

Reconstructing Onions The fact that a random padding is used prevents the re-construction of previous onions. The same is true for the data messages through the virtual circuit. They have a fixed size and previous messages cannot be reconstructed because the symmetric cryptographic functions and keys are only known by the node processing that onion.

Decreasing Onion Size The simple padding mechanism ensures that all onions have the same length and so no information is revealed from the size of onions.

Fingerprint Attacks In the design there is no protection against this kind of attacks. But the fixed size of messages complicated this kind of attacks in practise.

End-to-End Attacks The design is very vulnerable to this kind of attacks because of the real-time character of the communication.

4.4.2 Active Attacks

Compromised First Node With designing the begin of a route as a proxy it is assumed that the first proxy node in the route can be trusted, because it has to know the destination. The most secure variant would be that the proxy is running on the same computer than the initiator's software, as already discussed.

Of course this computer can also be compromised, but in this case anonymity is broken anyhow, because an attacker can record all user actions.

Evil Nodes In this design it is not dealt with this question because the nodes are build into the software and are assumed to be trusted.

Repetitive Attack This attack is not possible for onions here. For this onions include the expiry time. For messages encrypted with symmetric cryptography it depends in the way how it is implemented. If the same inputs also leads to the same output a replay attack would be possible. Using an encryption scheme such as counter mode prevents this attack.

Iterated Compromise This is possible since the scheme used to encode the route consists of an onion which could be decrypted with all private keys of nodes inside this route.

4.4.3 Directory Attacks

In this design all nodes are build into the software because it is only intended as a prototype. So there is no directory distribution system.

5 TOR

TOR was designed by Dingledine et al. [7]. It is not only implemented as a prototype but it is free software and there is an open network which can be used by everyone to communicate anonymously [19]. Consequently it is under continuous development. In this section not only the original design but also changed to the protocol as far as they are relevant for anonymity are discussed.

5.1 General Design

Because TOR should be used in the real world the design also focuses on usability. To make it work with much software without the need to modify it, it uses a SOCKS proxy which works for almost each TCP/IP based protocol. Also many non-expert users should use the system. This is because more users also means more anonymity. Since better usability means more users, usability is called a security requirement in the design. In another paper [6] the designers explain in detail why usability is that important for anonymising networks. The usability of TOR was analysed in 2007 [3] and it was shown

that still it is very difficult to configure TOR in a good way for novice users. This can also reduce the anonymity if they configure things wrongly.

Also nodes should be run by different persons. So everyone should have the possibility to run servers. For this a mechanism is needed to distribute the list of servers in a secure way. Also server operators might want to have the possibility to limit the bandwidth used by TOR.

Misusing the TOR network for illegal or socially disapproved goals is also a possible threat to the whole network, because it can lead to server operators getting into trouble or to a lower the reputation of the whole network. This finally leads to less participants and consequently to less anonymity. This might not be the goal of people misusing the network. Most of them would misuse the network to do illegal actions anonymously. To reduce the cases of abuse server operators can restrict the services her server can be used with. This can be done with so called *exit policies*. It is even possible to configure a server in such a way that is only used in the middle of a route but never connects to a service directly.

In 2008 a study was done for showing misuse of the TOR network [14]. In the experiment the network was misused by clients for breaking copyright laws or for hacking attempts. Also the network was misused by server operators to log user traffic. This is possible because there is no encryption between the exit node and the destinations if the user is not using an encrypted protocol. TOR was misused to collect passwords for insecure protocol such as HTTP or Telnet.

The general design is similar to the one discussed in Section 4. The first node acts as a proxy, a virtual circuit is constructed and messages are send along them to the last node which connects to the responder. But there are some differences, discussed below, especially how the circuits are created.

5.2 Creating Circuits

There is not one onion containing the whole route but the route is constructed incrementally by the first node. So the route is expanded one-by-one using the already existing part of the route to relay messages to the node that is added. The first node negotiates a symmetric key, which is used for doing 128-bit AES cryptography in counter mode, with each node in a Diffie-Hellman [5] like way. The protocol looks like this:

$$Alice \rightarrow Bob : \{g^x\}_{E_{Bob}} \tag{16}$$

$$Bob \rightarrow Alice : g^y, H(K|"handshake") \tag{17}$$

Here Alice is the first node and Bob the node added to the circuit. E_{Bob} is Bob's public key, $K = g^{xy}$ is the negotiated key and | means concatenation. With the hash Bob authenticates to Alice because to construct the key he has to be able to decrypt the first message encrypted with his public key. In this way there can be no man-in-the-middle attack.

With this protocol it is achieved that the key is only known by Alice and Bob and it is not possible to retrieve the key from the messages send, even if the private key of Bob is known. This is called *perfect forward secrecy* and prevents that a route can be revealed by compromising the node's keys one by one, as discussed in Section 5.7.

To limit the time an attacker has to compromise one circuit, periodically each nodes closes them if they are not used any more and build new ones.

5.3 Sending Messages

Sending messages works in a way similar to the original design. Messages are used to create and destroy circuits and to send data. But there are some differences.

For all connections end-to-end integrity check is done. For this a 48-bit *SHA-1 digest* [8] is used which is computed by the first node and checked by the last one. For all nodes in between the digest is meaningless because it is encrypted. The reasons for having integrity checking are that an internal attacker, who can guess the content of messages, can change the content of the messages and therefore for example change the destination of a message.

Another difference is that TOR has a *leaky-pipe circuit topology*. This means that the message can leave the circuit at any point and not only at the last node. This can for example be necessary because of different exit policies of the nodes. Also this can make some kinds of attacks more difficult as discussed later. If a node received a valid digest the message is processed by this node, otherwise it is send to the next node in the circuit. If the last node receives a message with an invalid digest the

circuit is closed immediately which makes it impossible for an attacker to guess correct digests.

5.4 Distributing the Directory

Nodes of the system have to know about all other nodes to be able to pick a route randomly. Distributing the list of servers (called *directory*) is a crucial and also security relevant point of the system. There are many attacks on the directory breaking the anonymity as will be discussed later in more detail.

In TOR three different versions for the directory system were used, because of problems with the previous ones. In the following the differences between those versions are explained. Detailed specifications can be found in the specification directory of TOR's repository [21].

5.4.1 Version 1

In the original design the directory is provided by a small number of trusted *directory servers*. A list of all server and their keys is preloaded into the node's software. Each node can load the signed directory from a server. Because signatures are used the directories can also be cached.

Nodes send signed state information (called *descriptors*) to the directory servers. Before a node is actually added to the directory the administrator has to manually include it. This prevents an attacker to add a large number of nodes at the same time. The directory servers have to synchronize, because differences in the directories could assist an attacker.

5.4.2 Version 2

There are several problems with the original design. One problem is that an increasing number of nodes causes more traffic and therefore bandwidth problems for the server. One reason for this is that the directory has to include all nodes of the network. Adding more directory servers increases synchronization problems and is therefore no solution. So the old design was not scalable for an increasing number of nodes. Also each directory server is a trust bottleneck because it can provide a number of clients with wrong information for some time which can be used to attack anonymity.

So a second version for directory distribution is used for TOR 0.1.1.x and 0.1.2.x. To save bandwidth *network status documents*, which does only include a list of the nodes with only a hash of the descriptor, are used. Each client can decide of which node it does not have the recent descriptor and download only the changed ones. There is also a number of directory servers but they are only semi-trusted. They do not synchronize but provide the network status documents of many servers. Each client accepts a descriptor as valid if it is included in the network status documents of half of the directory servers. For this they need a minimal number of documents but not the documents of all directory servers, before circuits can be build. Another difference is that each node can act as directory mirror and provide a number of network status documents. If they are outdated, the documents are not used any more and requested again.

5.4.3 Version 3

In this version it is returned to a system were the authority servers agree on a single network status document instead of letting the clients decide based on a number of different network status documents. This is done because this can lead to clients with different knowledge of the network which can help attackers. Also the keys of the directory servers are protected better by using signing keys which are certified by the server's keys. So the keys do not have to be stored on the servers.

An open problem is that with some network size it is not practicable that each node knows each other one any more.

5.5 Hidden Services

TOR allows to offer a TCP server without leaking the IP address to people using that service. This corresponds to *SR1*.

In TOR so called *rendezvous points* are used for this. If Bob wants to of offer a service he chooses some nodes to act as *introduction points* for his service. So the service does not rely on a single node making it more robust. To identify his service he uses a public key pair. The public key is sent to all introduction points. Then he advertises his service, the corresponding public key and the list of introduction points to a lookup service us-

ing a pseudomyn for his service. This information is signed by the public key identifying the service such that it cannot be manipulated by anyone not knowing the service's private key. The pseudonym bounded to a public key gives his service a long-term pseudonymous identity.

If Alice wants to use the service she looks it up in the lookup service, chooses one node of the network as rendezvous point and informs one of Bob's introduction points about that. For this she gives a randomly chosen *rendezvous cookie* and the first half of the handshake to it. Bob is now informed about this by the entry point and can now decide if he want to make a connection with Alice which gives him the possibility to avoid denial-of-service attacks. If he wants Alice to make a connection he send the rendezvous cookie together with the second half of the handshake to Alice's rendezvous point. The rendezvous point does not know the identity of Alice or Bob since all communication is done via the TOR network. It can now connect Alice's and Bob's circuits such that Alice can send a request to Bob's service.

5.6 Changes to the TOR design

TOR is a project which is under constant development. Here the changes to the protocol retrieved from the project's release notes [20] which are relevant for the security are given. Most security relevant improvements are about implementation issues, for instance buffer overflows or wrong behaviour for malicious input, issues were the implementation did not fit to the specification or parameters which could be security relevant, for example how often to rotate some keys.

Apart from changes to the directory protocol which are already described earlier there was one major security relevant change to the protocol which is the introduction of entry guards. They are specified in the *Tor Path Specification document* in the specification directory [21]. The goal is to reduce the likelihood that an attacker controls both the entry point and the end point of a route. In this situation it is possible to carry out statistical attacks.

If an attacker controls C out of N servers the probability that she controls both the entry and the exit point is $(\frac{C}{N})^2$. The problem is that if different nodes are chosen at random over a long period of time the chance that the attacker will have a sample of $(\frac{C}{N})^2$ of the traffic goes to 1.

The solution is to use a number of fixed nodes as entry points. Because nodes can be temporary unavailable an ordered list is maintained which is stored persistently. For each connection one of the first nodes which is working at the moment is chosen. If there are too less available nodes new nodes are added at the end without removing the first ones. Nodes are only removed if they are not available for a long period of time. With this approach the chance that the entry point is bad is decreased. In the ideal situation were always the same node is used the probability is $\frac{C}{N}$.

5.7 Vulnerabilities

Here for the attacks described earlier it is discussed how likely they are with the TOR network. Also some attacks especially developed for the TOR network are given.

5.7.1 Passive Attacks

DNS Information Leakage The fact that a socks proxy is used for all applications leads to a vulnerability which is not directly related to TOR. The problem is that some application do not give the host-name to the proxy but first resolve it to get the IP address. This is done by connecting to a DNS server. But if this connection is not done via the TOR network, anonymity is compromised because the host-name is transfered in plaintext to the DNS server and it is obvious that someone who gets the IP address of someone else also wants to communicate with her.

Reconstructing Onions This attack is not possible because of the random padding used.

Decreasing Onion Size This attack is not possible because cells have a fixed length.

Fingerprint Attacks There are some characteristics of TOR which makes such an attack difficult. The first one is that streams are multiplexed within the same circuit. If the network is heavily used it is very difficult to tell which messages belongs to the same connection. Also the the fixed cell length complicates the analysis.

End-to-End Attacks The design is not completely immune to them as also stated in the design article. If users are running their own nodes this can make the analysis more difficult because not all data that is sent is related to the user but could also be caused to other ones. How much this helps depends on the usage of the network.But the network is never completely immune to such attacks, if an attacker gathers a sufficient amount of information.

5.7.2 Active Attacks

Evil Onion Routes This attack can be a serious problem for the network because it relies on many different people running nodes. Because of this an attacker can try to control a large number of nodes. In practise with a growing number of nodes it becomes very difficult for an attacker to control a significant number of nodes. Also in the directory system there are countermeasures to prevent attackers from adding a large number of nodes at the same time.

Repetitive Attacks Replay attacks are not possible any more because if during the handshake a message is send twice it will result in a different answer each time. For communication using symmetric cryptography in counter mode replay attacks are also not possible.

Iterated Compromise The attack to compromise the route one-by-one is not possible any more, because of perfect forward secrecy that is provided by the protocol. Because the symmetric keys are created using a handshake protocol, they are only known by the two involved nodes and can be deleted after the circuit is closed. There is no way reconstructing them.

Compromising Directory Servers By compromising directory servers it is possible to influence the directory partially. But because including a node into the directory requires the directory servers to agree, a significant number of directory server has to be controlled in order to be able to have a large impact on the directory accepted by the clients.

But at least with the first version of the protocol a single server could manipulate the directory for some client for a short period of time.

Measuring Node Loads It is possible to measure the load of a TOR node by using the latency of connections through this node. This is because different connections interfere with each other. Since TOR is designed for low latency no large random delays can be added and there is a correlation between a node's load and a larger latency for connections.

An attack developed by Murdoch and Danezis [15] uses a single malicious TOR node to measure theses latencies by making connections to certain nodes. So the attacker does not needed to be able to observer network traffic but gets the information only based on latencies for connections made to other nodes. In their attack they want to reveal the identity of users connecting to the attackers service. This means breaking *SR2*. For this they send data with a special pattern (for example in short bursts) to the client. Then the attacker can observe the latency of nodes to see if there is a correlation and can eventually reveal the user's entry point. The attacks turned out to be very effective in 2004 but with an increasing number of nodes it becomes more difficult to perform the attack since the load of a larger number of nodes has to be measured to find one for with the load correlates to the traffic pattern sent.

5.7.3 Attacks on Hidden Services

For breaking the anonymity of hidden services basically the same attacks are possible than for each communication via the TOR network. Additionally there are denial-of-service attacks. Also it would be very bad if someone could replace a hidden service. But since this is not directly connected to anonymity it is not analysed in detail here.

6 Universal Re-Encryption

Universal re-encryption [11] is an interesting technique which can be used for onion routing. In [12] a scheme is proposed for which the authors claim that it is immune against repetitive attacks. It makes it

also easy to find attackers disturbing the communication.

Also with universal re-encryption it is possible to make a design where the route does not have to be determined by the first node but can be determined by trusted servers or can dynamically be extended [13].

6.1 Basics of Universal Re-Encryption

Universal re-encryption is based on the El-Gamal encryption scheme [9]. This scheme has the interesting property that the cyphertext of the same plaintext with the same key yields different results every time. Even stronger it is not possible to tell that two cyphertexts were created with the same key if the private one is not known. The interesting property for re-encryption is that everyone can re-encrypt a cyphertext. This means making another cyphertext representing the same plaintext without decrypting the messages. But for this the public key is required with the original scheme.

In contrast universal re-encryption can be done without the knowledge of the public key. This is done in this way. First the same things as for El-Gamal are needed: a cyclic group G, a generator g, a random private key $x < p-1$ and the corresponding public key $y = g^x$. For universal re-encryption the ciphertext actually consists of two El-Gamal ciphertext: the one of the message m and the ciphertext of 1. So the ciphertext of m is:

$$URE_x(m) = (\alpha_0, \beta_0, \alpha_1, \beta_1) \quad (18)$$
$$= (m * y^{k_0}, g^{k_0}, 1 * y^{k_1}, g^{k_1}) \quad (19)$$

Here k_0 and k_1 are random numbers.

The plaintext can be retrieved by calculating α_0/β_0^x while α_1/β_1^x is always 1 but is needed for the re-encryption.

The re-encryption can simply be done by calculating (again k_0' and k_1' are random):

$$(\alpha_0 * \alpha_1^{k_0'}, \beta_0 * \beta_1^{k_0'}, \alpha_1^{k_1'}, \beta_1^{k_1'}) \quad (20)$$

Because k_0' and k_1' are random the result is different for re-encrypting the same message multiple times, but still the plaintext after decryption is the same.

It is also possible to additionally encrypt a message with an arbitrary new private key x' and corresponding public key $y' = g^{x'}$ without knowing the original private key:

$$URE_{y+y'}(m) = \quad (21)$$
$$(m * y^{k_0} * y'^{k_0}, g^{k_0}, 1 * y^{k_1} * y'^{k_1}, g^{k_1}) \quad (22)$$

6.2 Onions Based on Universal Re-Encryption

To be useful for onion routing this scheme can be used in the following way. Each server has a private key x_i and public key y_i. For a route of servers $1, \ldots, n$ a message m is encrypted in the following way:

$$URE_{x_1,x_2,\ldots,x_n}(m) = (\alpha_0, \beta_0, \alpha_1, \beta_1) = \quad (23)$$
$$(m * (y_1 \cdots y_n)^{k_0}, g^{k_0}, 1 * (y_1 \cdots y_n)^{k_1}, g^{k_1}) \quad (24)$$

Each server s can partially decrypt $URE_{x_s,x_t,\ldots,x_n}(m)$ by computing:

$$URE_{x_t,\ldots,x_n}(m) = (\alpha_0/\beta_0^{x_s}, \beta_0, \alpha_1/\beta_1^{x_s}, \beta_1) \quad (25)$$

Then the retrieved new cyphertext can be re-encrypted as described above and send to the next server.

The content of such a block is only readable by the last node. For server s this is done with the key $x_1, x_2, \ldots, x_s$. So for encoding a route one block for each server must be send. This means that an onion consists of a number of blocks. The content of the blocks is the next hop of the route. Only the block for the last node contains the message. So an onion looks like this:

$$URE_{x_1}(\text{"to } S_2\text{"}), URE_{x_1,x_2}(\text{"to } S_3\text{"}), \ldots, \quad (26)$$
$$URE_{x_1,\ldots,x_{n-1}}(\text{"to } S_n\text{"}), URE_{x_1,\ldots,x_n}(m) \quad (27)$$

Each node must partially decrypt and re-encode all onions and send them to the next server. In the proposed protocol the order of the onions is even permutation randomly.

6.3 Prevention Against Repetitive Attacks

With this scheme each time an evil server replays an old onion, the generated onion will be different. So the attacker cannot observer equal messages send be the attacked server.

But there is another attack called *multiplicative attack*. Assume that an adversary controls at least two server on a route. The first server processes a number of blocks $URE_{k_i}(m_i)$. Assume that $URE_{k_S}(m_S) = (\alpha_0, \beta_0, \alpha_1, \beta_1)$ is readable by the second server S which is under the control of the adversary and the first server replaces α_0 by $\alpha_0 * \gamma$ where γ is chosen arbitraryly. Server S receives $m_S * \gamma$ instead of m_S and if it knows about γ it can reconstruct m_S and carry on with the protocol.

Of course the first server can only guess if there is an onion for a second server under her control. So the attack consists of trying to manipulate blocks. If the block is not intended for an adversary's server, at one point the address of the next node is invalid and the message does not reach the destination. So by trying to manipulate blocks randomly an adversary can get some information about the route, but this attacks is not as bad as a repetitive attack, because the adversary has to guess which of the blocks to manipulate.

If the wrong block is manipulated one of the servers will discover that there is no valid address for the next node. In this case it knows that some of the previous servers is dishonest. A protocol is proposed for finding out the dishonest server which is then rejected from the protocol. The protocol for finding out the dishonest server is based on a proof each server has to do. It is possible to proof that the re-encryption was done in a good way without revealing the private key by showing the equality of some logarithms.

Onions based on universal re-encryption are protected again repetitive attacks at least in the original form, but as discussed later a similar attack is possible. The advantage compared to the handshake based solution in TOR which also provides protected against those attacks is that no bidirectional communication is needed and there is a lower latency. However for real-time services where bidirectional communication is needed anyhow there is not a problem with TOR's solution since only when opening a circuit the handshake protocol is needed. For the actual data also universal re-encryption is not useful since it is very inefficient compared to symmetric cryptography.

6.4 On-line and Off-line Onion Encoding

In all previous described designs the first node chooses the whole route. The disadvantage is that the initiator has to know the whole network topology, which leads to the difficulty with distributing a directory, especially for dynamic networks.

In [13] two more dynamic ways are given based on universal re-encoding: *off-line* and *on-line onion encoding*.

6.4.1 Off-line Onion Encoding

The idea of off-line encoding is that a *navigator* can be prepared by a specialized server. This navigator already includes the route. A user can include the message into it and send it via the network without knowing the actual route.

Message Insertion With universal re-encryption it is possible to prepare a ciphertext into which a message m can be included later. This is done by first preparing the ciphertext of 1:

$$(1 * y^{k_0}, g^{k_0}, 1 * y^{k_1}, g^{k_1}) \tag{28}$$

To retrieve the ciphertext of m the first field is multiplied by m. To do this no knowledge about the key is needed.

Navigators A navigator is an onion which encodes a special void message. As described before an onion consists of a number of blocks. A navigator to a certain node N is created in such a way that N retrieves the special void message as plaintext and so knows that it is the last node in the route. A navigator from node A to B is denoted as $Nav[A, B]$. Additionally to an navigator a ciphertext containing 1 with decryption key x is used when the onion is created. The whole onion looks like this: $Nav[A, B], URE_x(1)$. An arbitrary message m can be inserted by a client as described earlier. So the whole onion becomes $Nav[A, B], URE_x(m)$ and can be used to route m via a route from A to B.

Advantages The advantage of this is that clients do not need to know anything about the network topology any more. So the chance that a traffic analysis becomes possible because of different

knowledge of clients is lowered. Also a navigator can be used as anonymous return address. For example this can be used to achieve *SR1*. The advantage compared to for example rendezvous points of TOR is that the servers do not have to remember anything. The disadvantage of this scheme is that there has to be a single trusted server which has to know the whole network topology. But in fact this is the same situation than with the TOR network, where a user has to trust the first node and each node has to now the whole dictionary.

6.4.2 Merging Navigators

To solve the problem that a user has to trust a single server a scheme is proposed for merging navigators from different sources to a new one, which encodes a longer route.

The two navigators $Nav[A, B]$ and $Nav[C, D]$ can be merged for sending m to Bob in this way:

$$Nav[A, B], URE_{x_A}(C), URE_{x_A}(Nav[C, D]), \quad (29)$$
$$URE_{x_A+x_B}(Bob), URE_{x_A+x_B+x_{Bob}}(m) \quad (30)$$

6.4.3 On-line Merge Onions

For this scheme the sender has to know at least some nodes of the network. The sender chooses a route of nodes through which the messages is send and encodes it with a navigator, but the communication between the nodes of this route is not done directly but also via the network via a route with arbitrary nodes, which are freely chosen by the nodes in the sender's original list.

The advantage of this is that the length of paths can be adopted on-the-fly depended on the load of the network. This can make differences in network load smaller complicating traffic analysis. Also it reduces the traffic for long routes because the number of initial nodes in the route can be small compared to the actual length of the route.

6.5 Vulnerabilities

Although the solutions given in this section should prevent some kinds of attacks it is possible to break them as schown by Danezis [4].

6.5.1 Repetitive Attack

Although Gomulkiewicz et al. [12] claim that there design is immune against repetitive attacks, a simply scheme for doing such an attack is introduced by Danezis [4]. For this a label is attached to the onion by a malicious node. The label L is encrypted in that way:

$$URE_{y_i+y_m}(L) \quad (31)$$

The output of the next node will contain:

$$URE_{y_m}(L) \quad (32)$$

This is readable by the attacker who has the private key belonging to y_m. So the attacker has the same information as with the normal repetitive attack.

6.5.2 Replace Blocks

Another attack described by Danezis [4] works by replacing the blocks of an onion. Each block $URE_{x_1,\ldots,x_n}(\text{"to } S_{n+1}\text{"})$ is replaced with:

$$URE_{x_1,\ldots,x_n}(\text{"to } Eve\text{"}), \quad (33)$$
$$URE_{x_1,\ldots,x_n+x_{Eve}}(\text{"to } S_{n+1}\text{"}) \quad (34)$$

Node S_n processes the onion and sends it to Eve since the partially decrypted onion contains this as next destination. The onion send to Eve contains this for the next node of the route:

$$URE_{x_{Eve}}(\text{"to } S_{n+1}\text{"}) \quad (35)$$

Eve can decrypt that and retrieve the next node of the route in this way. This attack can then be repeated for each node of the route until it is entirely revealed. The resulting route is illustrated in Figure 3. The message is routed from Eve to each node of the original route and then back to Eve.

7 Anonymity Provided By Onion Routing

Here it is evaluated what kinds of anonymity can be achieved by onion routing and how strong this anonymity is. Also open problems are discussed.

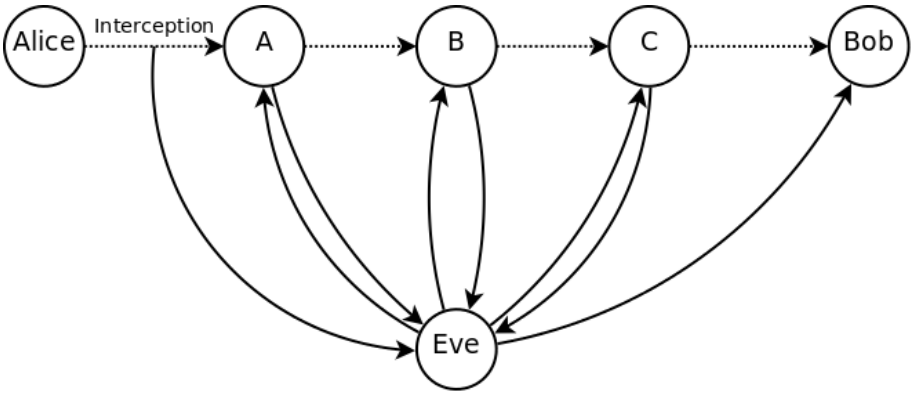

Figure 3: Attack on Universal Re-Encryption

7.1 Evaluation of Security Requirements

For all security requirements given in Section 2.3 it is evaluated how good it is achieved by solutions examined in this paper and if it can be achieved by onion routing in general.

SR1 If Bob wants to provide an anonymous service that can be achieved by using TOR's hidden services. It can also be achieved by a solution based on universal re-encryption. This requirement is weaker than *SR3*. If an external attacker can get to know to whom Alice is communicating, Alice can also get to know.

SR2 This security requirement can be achieved by all onion routing solutions. Again there is a strong related security requirement, which is *SR4*. If an external attacker can get to know to whom Bob is communicating Bob can also get to know. Of course, it is possible that *SR4* holds but Bob knows the identity of Alice because an authentication protocol is used on top of the onion connection.

SR3 and SR4 There is no essential difference between those two properties from the point of view of an attacker. It is about getting to know with which other parties someone communicated through the onion network. Who set up the connection does not really matter.

This is the main goal of onion routing and consequently all solutions tries to provide this. In practise there are many possible attacks to break this, but with a heavily used network as TOR the anonymity provided is reasonable. Still solutions are improved and the level of security in respect to these requirements is becoming higher.

SR5 Onion routing makes the confirmation difficult but with some effort it is possible to confirm if two parties are communicating by using a timing attack. Statistical attacks are a fundamental problem of onion routing and it is impossible to achieve that with the concepts used now. There seems to be a fundamental trade-off between the latency provided by an anonymity network and the likelihood that timing attacks can be performed.

SR6 None of the solutions analysed in this paper provides steganography.

7.2 Open Problems

One of the biggest problems of low-latency networks is that they leak much information by correlations between packets sent and the load of nodes. At the moment there seems to be a trade-off between achieving a low-latency for interactive web services such as the WWW and hiding information by introducing extra delays and reorder packets.

Further with an increasing network it is difficult to handle the directory. The first problem is that it becomes infeasibly for each client to download the whole directory, but on the other hand different knowledge among the clients is a threat for anonymity. The on-line encoding scheme provided

by universal re-encryption could be a possible solution, but it still has many problems and solutions based on this seems not to be very mature yet. The second problem is that it becomes too much work to manually remove misbehaving nodes from the directory and automatic methods to test of a node behaves in a good way are needed.

One last very important point is that the network has to be used by non experts. Such a network is only useful when it is actually used to provide people with anonymity and also the anonymity of each user increases with increased usage of the network. The fact that each user should run an own node to make end-to-end attacks difficult complicates the installation. There has to be a user-friendly solution to use such a network in a good way.

8 Conclusions

Onion routing provides no perfect anonymity. There are many kinds of active attacks for which no solution can provide real protection. Also a timing attack to verify whether two parties are communication is always possible in a low latency network.

At the moment TOR is the only mature solution which also has a network that is large enough to provide a reasonable level of anonimity.

To judge the level of anonymity it is important to take the intention of the attacker into account because it has to be evaluated if the effort is worth the benefit for an attacker. Onion routing provides protection against adversaries who what to gather a large amount of information by doing traffic analysis for example for commercial reasons. For such adversaries there is no benefit of getting only communication data of a small number of persons and attacking a large number of parties costs too much effort. But if the goal is to really hide with whom one is communicating it is always possible to reveal hat.

The risk has also to be compared with other possible ways to break anonymity, for instance by installing malicious software on the computer. A personal computer is an insecure environment anyhow and there is a chance that an attacker can get to know all communication that is done with it.

References

[1] The anonymizer. http://www.anonymizer.com/.

[2] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 1(1):84–88, February 1981.

[3] Jeremy Clark, P. C. van Oorschot, and Carlisle Adams. Usability of anonymous web browsing: an examination of Tor interfaces and deployability. In *Proceedings of the 3rd Symposium on Usable Privacy and Security (SOUPS '07)*, pages 41–51, New York, NY, USA, July 2007. ACM.

[4] George Danezis. Breaking four mix-related schemes based on universal re-encryption. In *Proceedings of Information Security Conference 2006*. Springer-Verlag, September 2006.

[5] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory IT*, pages 644–654, November 1976.

[6] Roger Dingledine and Nick Mathewson. Anonymity loves company: Usability and the network effect. In Ross Anderson, editor, *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)*, Cambridge, UK, June 2006.

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[8] D. Eastlake 3rd and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174 (Informational), September 2001. Updated by RFC 4634.

[9] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[10] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. Hiding Routing Information. In R. Anderson, editor, *Proceedings of Information Hiding: First International Workshop*, pages 137–150. Springer-Verlag, LNCS 1174, May 1996.

[11] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul Syverson. Universal re-encryption for mixnets. In *Proceedings of the 2004 RSA Conference, Cryptographer's track*, San Francisco, USA, February 2004.

[12] M Gomulkiewicz, M Klonowski, and M Kutylowski. Onions based on universal re-encryption - Anonymous communication immune against repetitive attack. In Lim, CH and Yung, M, editor, *INFORMATION SECURITY APPLICATIONS*, volume 3325 of *LECTURE NOTES IN COMPUTER SCIENCE*, pages 400–410. SPRINGER-VERLAG BERLIN, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2005.

[13] Marcin Gomulkiewicz, Marek Klonowski, and Miroslaw Kutylowski. Anonymous communication with on-line and off-line onion encoding. In *Proceedings of Workshop on Information Security Applications (WISA 2004)*, August 2004.

[14] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining light in dark places: Understanding the Tor network. In Nikita Borisov and Ian Goldberg, editors, *Proceedings of the Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 63–76, Leuven, Belgium, July 2008. Springer.

[15] Steven J. Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*. IEEE CS, May 2005.

[16] A. Pfitzmann, B. Pfitzmann, and M. Waidner. Isdn-mixes: Untraceable communication with very small bandwith overhead. *GI/ITG Conference on Communication in Distributed Systems*, pages 451–463, February 1991.

[17] Charles Rackoff and Daniel R. Simon. Cryptographic defense against traffic analysis. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 672–681, New York, NY, USA, 1993. ACM.

[18] Paul Syverson, Gene Tsudik, Michael Reed, and Carl Landwehr. Towards an Analysis of Onion Routing Security. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 96–114. Springer-Verlag, LNCS 2009, July 2000.

[19] Tor: anonymity online. http://www.torproject.org/.

[20] Tor release notes. https://git.torproject.org/checkout/tor/master/ReleaseNotes.

[21] Tor specification directory. https://git.torproject.org/checkout/tor/master/doc/spec/.